2019-04-08

Autonomous Vehicular Landings on the Deck of an Unmanned Surface Vehicle using Deep Reinforcement Learning

Polvara, R

10.1017/s0263574719000316
Robotica
Cambridge University Press (CUP)

Robotica (2013) © Cambridge University Press 2013
doi:10.1017/xxxx

Autonomous Vehicular Landings on the Deck of an Unmanned Surface Vehicle using Deep Reinforcement Learning

Riccardo Polvara‡*, Sanjay Sharma†, Jian Wan†, Andrew Manning† and Robert Sutton†

‡*Lincoln Centre for Autonomous Systems Research, School of Computer Science, University of Lincoln, Brayford Pool, Lincoln, LN6 7TS, UK.*
†*Autonomous Marine Systems Research Group, School of Engineering, University of Plymouth, Drake Circus, Plymouth, PL4 8AA, UK.*

(Accepted MONTH DAY, YEAR. First published online: MONTH DAY, YEAR)

SUMMARY
Autonomous landing on the deck of a boat or an unmanned surface vehicle (USV) is the minimum requirement for increasing the autonomy of water monitoring missions. This paper introduces an end-to-end control technique based on deep reinforcement learning for landing an unmanned aerial vehicle on a visual marker located on the deck of an USV. The solution proposed consists of a hierarchy of Deep Q-Networks (DQNs) used as high-level navigation policies that addresses the two phases of the flight: the marker detection and the descending manoeuvre. Few technical improvements have been proposed to stabilise the learning process, such as the combination of vanilla and double DQNs, and a partitioned buffer replay. Simulated studies proved the robustness of the proposed algorithm against different perturbations acting on the marine vessel. The performances obtained are comparable with a state-of-the-art method based on template matching.

KEYWORDS: Deep Reinforcement Learning; Unmanned Aerial Vehicle; Autonomous Agents;

1. Introduction

Over the recent years, the deployment of unmanned aerial vehicles (UAVS) in the marine environment has become a practice. The possibility to acquire images from a respectable height represents an undoubted advantage in search and rescue missions,[1] but also for water and coastal monitoring.

On the market there are multiple technologies related to UAVs, but those offering hovering and vertical take-off capabilities are the most interesting and easily deployable. The ability of landing autonomously is very important for UAVs, and achieving this on the deck of a un-/manned ship is still an open research area.[2] A UAV deployed in a marine environment is subject to wind and/or waves influences that can affect its manoeuvre.

The existing work in this field is based on hand-crafted features extraction and external sensors to identify the landing pad, usually represented by an high-contrast marker. In this paper a completely different solution is proposed, based on the recent interest born around the Deep Reinforcement Learning (DRL) and the Deep Q-Network (DQN) architecture.[3] It only uses low-resolution grey-scale images acquired by a down-looking camera and it outputs high-level control commands that lead the drone in proximity of the marker. The main advantage of this method compared to the existing ones is the

* Corresponding author. E-mail: rpolvara@lincoln.ac.uk

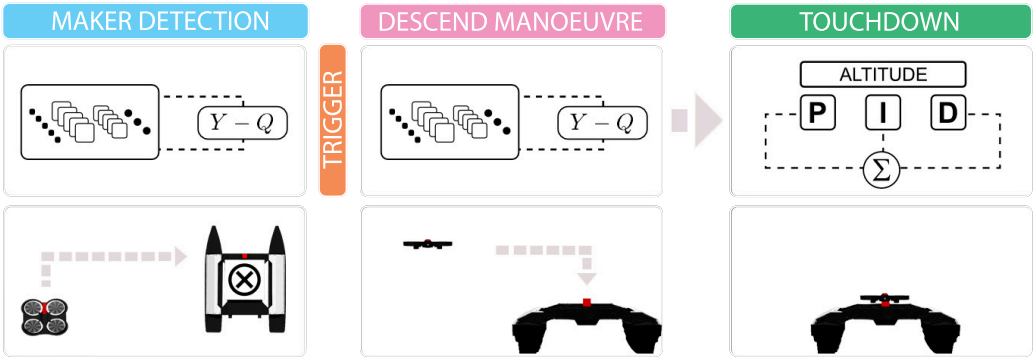

Fig. 1: Components of the proposed landing system. A first DQN takes care of the marker detection on the xy-plane. The second DQN handles the descending manoeuvre from 20 to 1.5 m. The last module is a closed loop controller that takes control from 1.5 m to the ground level.

total absence of any human intervention, allowing the quad-rotor to autonomously learn how to use high-level actions to achieve the task.

Previous DRL applications were limited only to deterministic environments such as the the Atari game suite.[3] Its applicability to the robotic domain is not straightforward and it has obtained only limited success so far, mainly related to the manipulation and grasping domain.[4] In this paper, a divide-and-conquer approach is adopted to split the landing task in two sub-tasks: landmark detection and vertical descent. Two specialized DQNs are responsible for addressing each phase of the flight and are connected through an internal trigger callable by the network itself. To solve the problem of sparse reward, the concept of partitioned buffer replay is introduced, splitting the experiences based on the reward values and guaranteeing the presence of rare transition at training time. An overview of the system is shown in Figure 1. To the knowledge of the authors, this work is the first unsupervised learning approach to tackle the landing problem on the deck of an unmanned surface vehicle.

In terms of the paper organisation, Section 2 presents the background of existing methods for autonomous landing, while Section 3 defines the problem in terms of reinforcement learning and introduces the solution proposed. In Section 4 the experiments conducted, each with a different kind of perturbation acting on the landing platform, are presented and discussed. Finally, concluding remarks are made in Section 5.

2. Related work

Autonomous landing is so far one of the most demanded features by UAV pilots. Despite this, it is still one of the most challenging aspects for this kind of vehicle. In order to compare the proposed method with the existing approaches, a brief background on the existing techniques is now offered. An exhaustive review is outside the scope of this paper so the reader is referred to recent surveys.[5,6]

Before describing the literature there are three aspects to take into account while landing: control, pose estimation, and navigation. Control represents the low level layer and consists of integrating the data from the sensors (GPS, IMU, altimeter, etc) in order to keep the vehicle around a desired set-point. The pose estimation is strictly related to control and consists in using sensors and visual feedback to estimate the position of the UAV with respect to a reference frame, which is the landing pad in this case. Finally the navigation: once the pose of the vehicle has been estimated, it is necessary to create and adjust a trajectory for reaching the target point. For the purpose of this paper, this section only focuses on literature concerning pose estimation and navigation.

In order to help the reader to understand the state of the art, the existing methods are grouped in three major areas: sensor-fusion, device-assisted, and vision-based. Combining different data from multiple sensors is a common practice used to improve the performances. This is the main strategy used in sensor-fusion systems. Similarly to Gomez-Balderas *et al.*,[7] Saripalli *et al.*[8] combined vision with differential GPS for estimating the relative pose of the UAV with respect to the helipad. In this work, geometric invariant extraction is used to identify a H-shape landing pad. The same technique was also applied in a successive work[8] in which the UAV landed on a moving target using a Kalman filter. Herisse *et al.*[9] combined the measurements of optical flow, gyroscopes and accelerometers to increase the robustness of a vision system for hovering and landing a flying vehicle on a moving target. Additional research based on optical flow for tracking and approaching a landing area has been undertaken by Cesetti *et al.*[10] and Ruffier *et al.*[11] There are two main problems in sensor-fusion approaches. First the sensor noise, which may be extremely high in an unstructured environment. Second, the unavailability of navigation information and/or improper sensor feedback. For instance in remote areas it is not possible to use GPS to adjust the trajectory.

On-board and ground devices have been widely adopted to increase the precision of marker identification. Kim *et al.*[12] used an omni-directional camera to enhance the field of view of an UAV and to identify a red marker. In Yakimenko *et al.*[13] an on-board infra-red camera was used to identify three known ground points and estimate their distance to the UAV. Infra-red ground spots are another expedient used to help the detection of the marker. For example, Wenzel *et al.*[14] used an algorithm for detection and tracking them with a downward looking camera. Other devices commonly used are distance measurement sensors, such as the Laser Imaging Detection and Ranging (LIDAR). LIDAR data was also fused with altitude measurements and feature-tracking data for estimating the UAV pose with respect to a fixed point in a selected landing area[15] . Similarly, Scherer *et al.*[16] used the LIDAR to identify and validate landing zones for a full-scale helicopter. The limits of these approaches are enumerable. First of all, the use of dedicated hardware both on the ground and on the vehicle is not always possible. Many commercial UAVs have limited hardware resources or limited payload capacity, which means that it is not feasible to use specific on-board devices. Second, some of these devices are expensive (e.g. LIDAR) and it is not possible to justify their use in commercial products.

Vision-based systems primary rely on camera images for pose estimation and planning. These methods identify the ground marker and extract relevant feature which allow for estimating the vehicle's pose. This data is then used within a control loop for approaching the pad. Sereewattana *et al.*[17] used Histogram of orient gradients (HOG) feature extraction and a pair of images taken at a known interval to estimate the distance between the vehicle and the marker. The adoption of a specific design for the pad (e.g. H-shape) allows performing corner detection and labelling after proper image binarization.[18] Shakernia *et al.*[19] address the vision problem as an ego-motion estimation one, estimating the altitude of the vehicle in relation to a fixed planar surface. In this way, vision is used in the feedback loop as a state observer for the landing controller. Vision-based systems tend to use only limited hardware for control and planning, this is advantageous for commercial vehicle which rely only on limited resources. However, most of these approaches are not particularly robust to image noise and may have problems when the pad is distant, partially occluded or blurred.

The methods previously described demonstrate different limitations and these are now discussed. Sensor-fusion methods rely on information coming from expensive sensors that most of the time are not present on-board of low-cost commercial UAVs. In addition, they mainly use a GPS signal that may be unavailable in real-world scenarios. The device-assisted methods usually provide an accurate estimate of the UAV's pose. However, they rely on external sensors that may be expensive and not always available. Finally, vision-based approaches use images acquired by on-board sensors, namely cameras, for control purposes. Even though this represents an invaluable advantage over other approaches,

they may fail when trying to identify low-level features in distorted images. The present work tries to solve the aforementioned problems. It relies only on a monocular on-board camera, without requiring any other sensor or external device. Image analysis is performed by the use of DQNs that significantly increase the robustness of the approach to transformations and corruption of the images. Here it is worthy to state that the intentions of the authors are not to prove that this work is better with respect to the state-of-the art. In contrast, the main point of the article is to address the landing problem from a completely different perspective and to show that it is feasible to develop an intelligent controller that does not require any human expertise.

The last part of this section is dedicated to reinforcement learning techniques that are related to the work here presented. The use of reinforcement learning in robotics is not novel, for example it has been widely used to achieve robust walking gait[20] and in manipulation tasks.[21] For a recent survey please refer to Kober *et al.*[22] However, works using reinforcement learning for drone control are quite limited. The main problem is related to an extremely large continuous control space and the unrealistic assumption of noise-free state-space; however, the present literature offers some successful applications and these will be discussed. For example, in Bagnell *et al.*[23] a policy search method has been used for controlling a small helicopter. In a similar way, Ng *et al.*[24] taught a mini helicopter to perform acrobatic manoeuvres, such as flying in a reverse status. Zhang *et al.*[25] combined reinforcement learning with model predictive control to train an obstacle avoidance policy which is allowed to access only sensor readings and not the full state of the system. In all these cases, the policy was intended to be a low level closed loop controller instead of the high level navigator considered in this work. The possibility to couple deep neural networks with reinforcement learning has recently opened new insight in a variety of domains. Since 2011, deep neural networks (DNNs) represent the state of the art in a variety of tasks, such as speech recognition,[26] head pose estimation,[27] and scene segmentation[28] (for a review see Schmidhuber *et al.*[29]). The use of DNNs as Q-function approximators allowed extending reinforcement learning to a new series of problems. A deep network has been used in Levine at al.[30] to learn hand-eye coordination for grasping in a robotic manipulator. Similarly,[4] a guided policy search method has been used to train a convolutional neural network for tasks that require close coordination between vision and control, such as screwing a cap onto a bottle. In Mnih *et al.*[31] an asynchronous variant of the actor-critic architecture obtained the best score in the Atari domain and applied to a variety of continuous motor control problems as well as on a new task of navigating random 3D mazes. Despite the recent effort in extending DRL to different domains, at the moment there is still no consolidated literature on drone control and planning.

3. Proposed method

In this section, the UAV landing issue is described in terms of reinforcement learning. Also, the technical solutions adopted are introduced and discussed.

3.1. Problem definition

As stated in Section 2, there is a limited amount of work trying to address the landing problem using reinforcement learning, in particular deep reinforcement learning. This is due to the fact the use of UAVs introduces many complications. In particular, aerial vehicles move within three-dimensional space which is time-consuming to explore, due to partial observability and occlusions. Moreover, the agent can see only a small portion of the environment and it has to deal with projective transformation. For this reason, applying DQN to the landing problem is challenging, even though it has already obtained state-of-the-art performances in two-dimensional games.

In this work, the landing problem is considered and divided into two sub-problems: the marker detection and the vertical descent. The first requires an exploration on the xy-plane, where the UAV has to align its body frame with the one of the marker located on the deck of the USV. In contrast, during the vertical descent phase, the UAV has to

reduce its altitude while keeping the marker centred. Since the introduction of a third spatial dimension leads to a larger state space and sparse reward, the vertical descent is the more challenging phase.

Two independent Deep Q-Networks have been trained separately for performing successfully the two tasks and integrated in a single state-machine. The first network has been trained to identify the marker at a fixed altitude of 20 m and guide the UAV above it. The DQN can stop the vehicle or move it in four directions (forward, backward, left, right). Moreover, it can call an internal trigger which activates the second network. The new policy can stop and move the vehicle in five direction (forward, backward, left, right and down). It aims to control the drone while vertically descending up to 1.5 m above the marker. The last module of the system is represented by a closed loop controller which slowly reduce the rotors power until the touchdown. The closed loop mechanism allows a fine control of the vehicle in the very last meter and can rely on distance sensors to safely land and turn off the UAV.

3.2. Notation

Formally, both the marker detection and the vertical descent problems can be modelled as Markov Decision Processes (MDPs). At each time step t the agent receives the state s_t, performs an action a_t sampled from the action space A, and receives a reward r_t given by a reward function $R(s_t, a_t)$. The action brings the agent to a new state s_{t+1} in accordance with the environmental transition model $T(s_{t+1}|s_t, a_t)$. In this study the transition model and the reward functions are not given (model free problem). The goal of the agent is to maximize the discounted cumulative reward called return $R = \sum_{k=0}^{\infty} \gamma^k r_{t+1}$, where γ is the discount factor. Given the current state, the agent can select an action from the internal policy $\pi = P(a|s)$.

In off-policy learning the prediction of the cumulative reward can be obtained through an action-value function $Q^{\pi}(s, a)$ adjusted during the learning phase in order to approximate $Q^*(s, a)$, the optimal action-value function. In simple cases Q can be represented in tabular form. The tabular approach is generally inadequate to model a large state space due to combinatorial explosion. To solve the problem, a function approximator (e.g. artificial neural networks) can be used to represent the action-value function. In this work, two deep neural networks are used as function approximators following the approach presented in Mnih *et al.*[3] The deep networks used take as input four 84×84 grey scale images acquired by the downward looking camera mounted on the drone, which are processed by three convolutional layers and two fully connected layers. The rectified linear unit[32] is used as activation function. The first convolution has 32 kernels of 4×4 with stride of 2, the second layer has 64 kernels of 4×4 with strides of 2, the third layer convolves 64 kernels of 3×3 with stride 1. The fourth layer is a fully connected layer of 512 units followed by the output layer which has a unit for each valid action (backward, right, forward, left, stop, descend and the trigger). Depending on the simulation, only a sub-set of the total actions is available (please refer to Section 4 for additional details). A graphical representation of the network is presented in Figure 2, while an example of the input images and the output of the first layer of kernel is provided in Figure 3.

The problem faced in DRL is to adjust the parameters θ, which are the weights of the network, minimizing a loss function $L(\theta)$ through stochastic gradient descend. The loss function used in this work is the following:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)} \left[\left(Y_i^{\mathrm{d}} - Q(s, a; \theta_i) \right)^2 \right] \tag{1}$$

where $D = (e_1, ..., e_t)$ is a dataset of experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ used to uniformly sample a batch at each iteration i. The network $Q(s, a; \theta_i)$ is used to estimate actions at runtime, whereas Y_i^{d} is the target that is defined as follows:

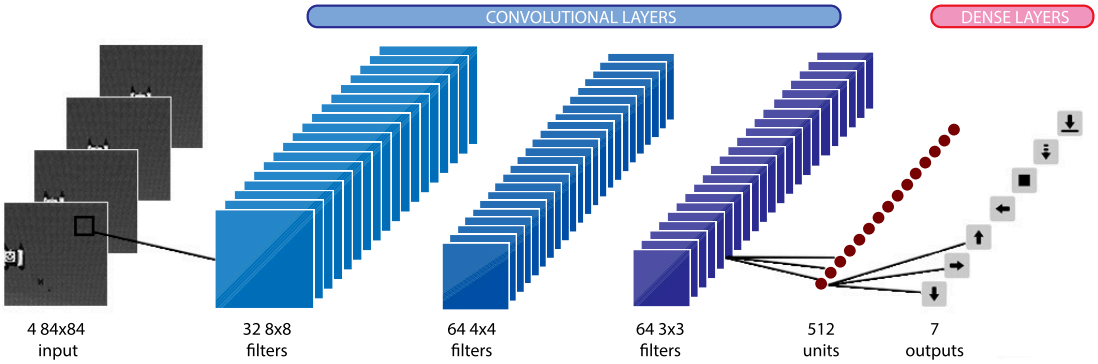

Fig. 2: Graphical representation of the DQN. The network takes in input four 84×84 images, and generates in output 7 actions: forward, right, backward, left, stop, descent, trigger.

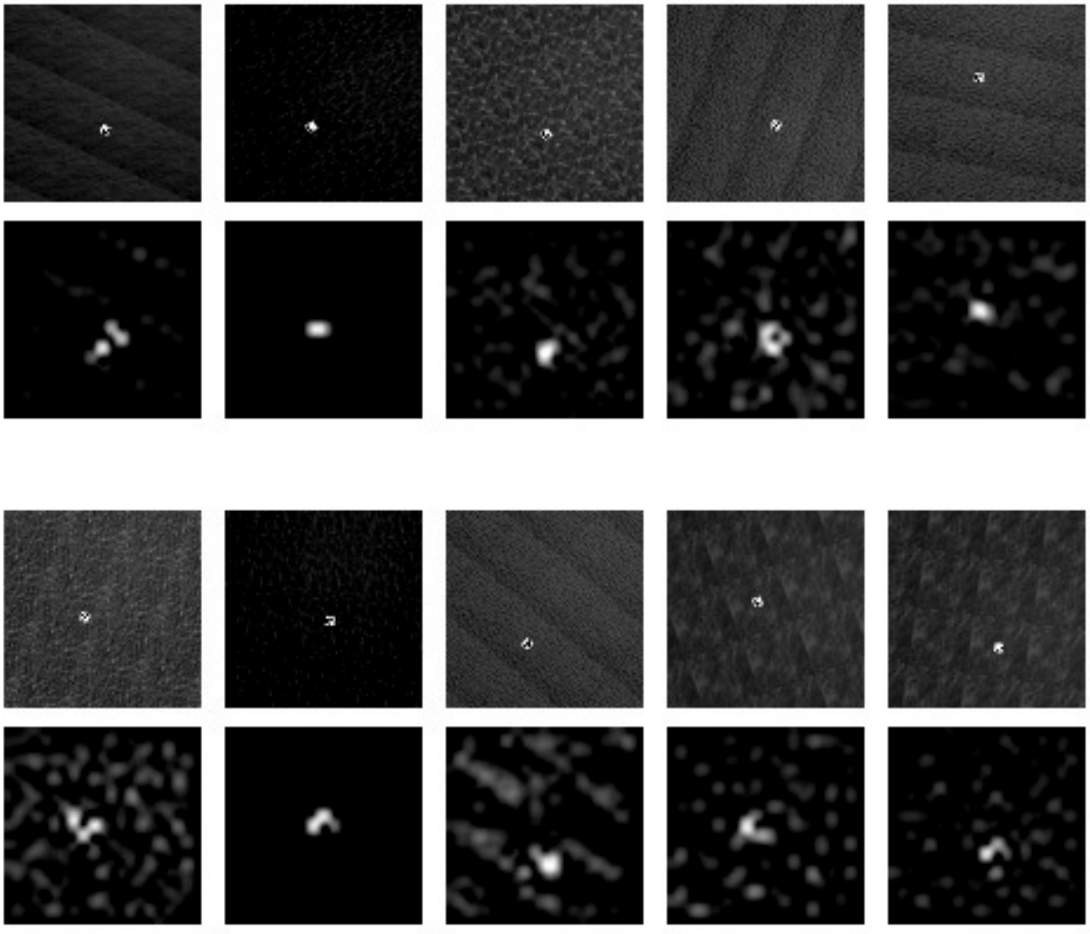

Fig. 3: The greyscale images given as input to the DQN and the feature map generated by the kernels in the first convolutional layer.

$$Y_i^{\mathrm{d}} = r + \gamma \; Q(s', \underset{a'}{\operatorname{argmax}} \; Q(s', a'; \theta_i); \theta_i^-) \tag{2}$$

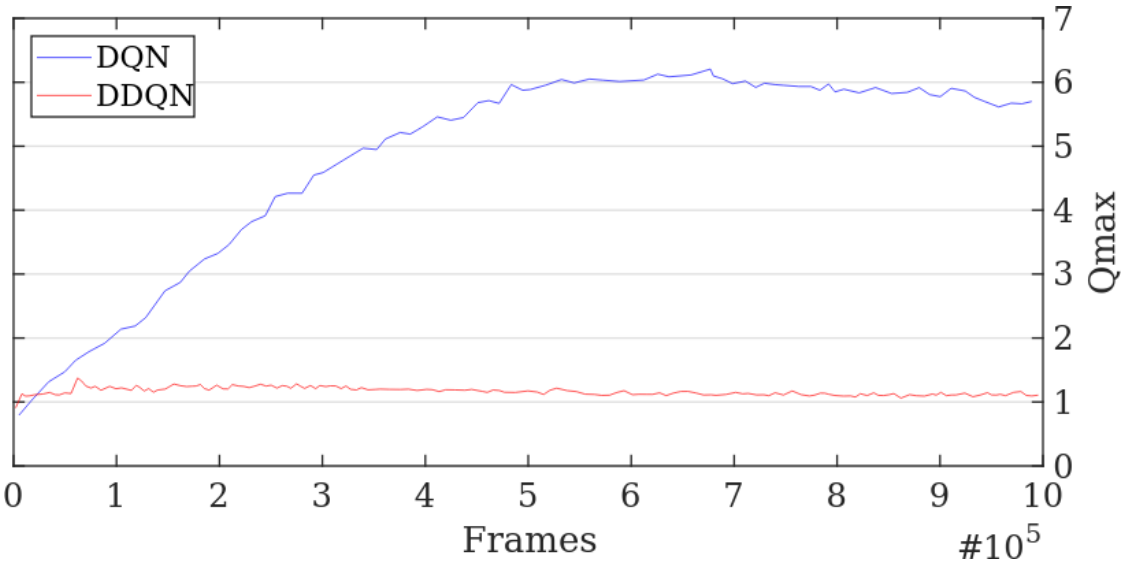

Fig. 4: Comparison between the Q-max value (the highest utility returned by the Q-network) obtained with the classic DQN and the revised Double DQN. The overestimation of the Q-function prevents the DQN to converge to the real maximum utility of 1.0 and learn how to accomplish the task.

The use of the target network is an expedient which improves the stability of the learning. The parameters θ are updated every C steps and synchronized with θ^-. In the standard approach, the experiences in the dataset D are collected in a preliminary phase using a random policy. The dataset D is also called *buffer replay* and it is a way to randomize the samples breaking the correlation and reducing the variance.[33] The target function used in this work is the one proposed in Double DQN approach[34] to solve the utility overestimation affecting Q-learning with large state space and sparse reward,[35] such as in this case. In fact, it was found that the *max* operator in the original work of Minh *et al.*[3] was responsible for the overestimation affecting all the action but the trigger. This is because the trigger, since it leads to a terminal state of the MDP, does not use the *max* operator for updating its utility. In a preliminary research it was experimented that the Q-max value (the highest utility returned by the Q-network) largely overshot the maximum utility of 1.0, which was correctly associated with the trigger (Figure 4). As a result, the drone moved on top of the marker and then randomly shifted on the xy-plane without engaging the trigger. Using the target function of eq. 2 solved the overestimation problem and the learning happened successfully.

Here it is particularly important to focus on the two phases that characterize the landing problem in order to isolate the obstacles that may be encountered. The landmark detection phase has a large state space. If the UAV is generated within a bounding box of size $15 \times 15 \times 20$ m and it has 20 time steps of approximately 1 meter per episode, it means that the agent has a box of $35 \times 35 \times 40$ m to explore. This huge volume can be reduced quite considerably, if it is assumed to fly at standard altitude. In this case the vertical alignment with the landmark can be done using only shifts on the xy-plane. This expedient does not have any impact on the operation level but it dramatically simplifies the task, reducing the volume to explore to a flat slice of size $35 \times 35 \times 1$ m. Additional details are given in Section 4. The vertical descend phase can be considered a form of Blind Cliffwalk[36] where the agent has to take a right action (descend) in order to progress through a sequence of n states. At the end of the walk, the agent can obtain either a positive or a negative reward. In the landing scenario, the structure of the problem makes extremely difficult to obtain a positive reward because the target-zone is only a small portion of the state space. The consequence is that the buffer replay does not contain enough positive experiences, making the policy unstable. To resolve this issue it is

possible to use a different form of buffer which encourage to replay important transitions more frequently, as proposed in Schaul *et al.*[36] A new type of buffer replay is therefore introduced and called partitioned buffer replay, that discriminates between rewards and guarantees a fair sampling between positive, negative and neutral experiences. Further details about the model implementation and the hyper-parameters used are discussed in Section 4.

3.3. Partitioned buffer replay

In MDPs with a sparse and delayed reward it is difficult to obtain positive feedback. In these cases, the experiences accumulated in the buffer replay may be extremely unbalanced. Neutral transitions are frequent and for this reason they are sampled with an high probability, whereas positive and negative experiences are uncommon and more difficult to sample. A preliminary study confirmed the vertical descent is affected by the sparsity of positive and negative rewards, leading to an underestimation of the utilities associated with the triggers. A solution for sparse reward was proposed by Narasimhan *et al.*,[37] dividing the experiences into two buckets depending on their priority (high or low). This work has been extended to K buckets. In Section 3.2 it has been defined $D = (e_1, ..., e_t)$ being a dataset of experiences $e = (s, a, r, s')$ used to uniformly sample a batch at each iteration i. To create a partitioned buffer replay, the reward space needs to be divided in K partitions:

$$R = R(s, a) \rightarrow \text{Im } R \ = R_1 \cup ... \cup R_K \tag{3}$$

For any experience e_i and its associated reward $r_i = r(e_i)$, it is possible to define the Kth buffer replay:

$$D_K = \{(e_1, ..., e_N) : r_1, .., r_N \in R_K\} \tag{4}$$

The batch used for training the policy is assembled picking experiences from each one of the K datasets with a certain fraction $\rho \in \{\rho_1, ..., \rho_K\}$.

In this work $K = 3$, meaning that there are three datasets with D^+ containing experiences having positive rewards, D^- containing experiences having negative rewards, and $D^\sim$ for experiences having neutral rewards. The fraction of experiences associated to each one of the dataset is defined as ρ^+, ρ^-, and $\rho^\sim$.

3.4. Hierarchy of DQNs

The solution proposed consists of a hierarchy of DQN to solve the different stages of the flight. In a finite-state machine fashion, the global policy is subdivided in modules each governed by a specific network which is also able to trigger the next one.

This approach is inspired by hierarchical reinforcement learning,[38] whereby sub-policies control the agents within a subset of a core MDP. In the present work, the overall landing task is mapped to the core MDP which is split into multiple simplified MDPs. In this way, standard Q-learning is still an effective algorithm for teaching the agent to learn how to solve the task.

Moreover, in this work it has been assumed that it is possible to combine the auxiliary MDPs in an ordered sequence and connect the elements of the sequence through shared states. In shared states the use of particular actions, called triggers, enables the passage to the next MDP. The triggers are additional actions that do not belong with the action space of the core MDP. Engaging the triggers in shared states leads to a reward that is equal to the maximal reward associated with the core MDP. How to define the auxiliary MDPs is an operation left to the designer that requires some knowledge of the core MDP.

The core MDP is therefore divided into three smaller consecutive problems: landmark detection, descent manoeuvre and touchdown. For the first two tasks, the networks have been trained to receive a positive reward when the trigger is called inside a target area. The last phase is outside the scope of this paper.

3.5. The simulated environment

Previous work in DRL was followed by the release of an open-source library for increasing the research in the field, such as the Arcade Learning Environment[39] including the Atari 2600 games and the OpenAI Gym[40] consisting in high level interfaces for classic problems (e.g. gridworld, pole balancing, mountain car, etc) and more complex ones (e.g. board games, humanoid walker, Doom the videogame, etc). Unfortunately, at the current stage there is no a standard environment for developing and testing UAV landing algorithms. To support the simulated experiments conducted, it has been worked on a simulated environment based on the Gazebo simulator[41] and ROS (Robot Operating System).[42] The simulator is called Quadrotor Landing Benchmark (QLaB) and is released under an open source license [1]. QLaB includes configuration files, textures and examples which allow the user to easily train and test new algorithms. The texture dataset consists of more than a hundred patches, 70% of which are being used for training and the remaining 30% for testing. The patches are high-definition pictures representing common ground floors belonging to eight categories: asphalt, brick, grass, pavement, sand, snow, soil and water. The *world* is a 100×100 m floor which can be covered by a uniform texture. Using a variety of floor it is possible to verify the generalization capabilities of the algorithm in different conditions. In this work only a small subset of all the textures available in QLaB has been used, in particular those relative to the marine environment. They represent different sea surface affected by light reflection and sea foam.

The UAV used in the environment is a widely diffused commercial quad-rotor, more precisely the Parrot AR-Drone 2. The vehicle has a weight of 420 grams, dimensions are 53×52 cm. It is equipped with two cameras, a frontal high-definition (1280×720) and a bottom QVGA (320×240). The hardware limitation allows for using only one camera at a time. The control command sent to the vehicle is represented by a continuous vector $\in [-1, 1]$, which allows moving the drone with a specific velocity on the three axis. The simulator allows training and testing in two cases:

1. Landmark detection. The vehicle is generated at a fixed altitude inside a bounding box of size $15 \times 15 \times 20$ m. To obtain a positive reward the drone has to find the landmark and moves over it in a target-zone of size $3 \times 3 \times 20$ m centred on the landmark.
2. Vertical descend. The drone is generated in a small bounding box of size $3 \times 3 \times 20$ m and it obtains the positive reward only when entering in a small target-zone of size $0.75 \times 0.75 \times 1.5$ m centred on the landmark.

To standardise the test, different parameters have been defined: number of attempts per texture, time limit, sensors available and marker type. For each one of the tests a total of 100 landing manoeuvres must be attempted. A time limit of 40 seconds (20 steps) is applied in the landmark detection test, whereas a time limit of 80 seconds (40 steps) is applied in the vertical descent test. When the episode finishes, a new one is started and the drone is randomly generated inside the large bounding box. The only sensors used is the downward looking camera, which can be read at a user-defined frame rate and resolution. Since there is not a standard marker, it has been decided to take the one used in the MBZIRC[43] competition which is a 1-meter white square containing a black circle and a cross. Finding this marker could be challenging because it does not contain any particular colour. Moreover, its shape could complicate features extraction due to the absence of well defined corners.

In the next section the primary contribution of this paper is introduced: the training and testing of a system for autonomous landing which is based on a hierarchy of DQNs.

4. Experiments

This section presents the methodology and the results obtained in training and testing the DQNs used in the proposed system.

[1] https://github.com/pulver22

4.1. First series of simulations
In the first series of simulations, the DQN responsible for the marker detection phase is trained on ten different textures resembling the sea surface. At each episode, the UAV started at a fixed altitude of 20m which was maintained for the entire flight. This expedient significantly reduces the state space for exploration while allows the marker to be visible during all the navigation. In order to stabilise the flight, long discrete movements were introduced, meaning each action is repeated for 2 seconds and then stopped, leading to an approximate shift of 1 m. The images from the camera are acquired between two consecutive actions, when the drone is moving at constant speed.

4.1.1. Methods. The training of the network is performed inside the QLaB simulator. The *world* is represented by a uniform marine texture of size 100×100 m with a fixed non-moving marker located at the centre (Figure 5). At the beginning of each episode, the drone was generated at an altitude of 20 m inside the perimeter of the large bounding box ($15 \times 15 \times 20$ m) with a random position and orientation (Figure 6). A positive reward of 1.0 was given when the vehicle landed in the target-zone, -1.0 otherwise. A negative cost of living of -0.01 was applied to all the other conditions. A time limit of 40 seconds (20 steps) was used to stop the episode and start a new one. The texture was modified every 50 episodes. The target and policy networks were synchronized every 30000 frames. The agent had five possible actions available: forward, backward, left, right and the trigger. The action was repeated for 2 seconds then the drone was stopped and a new action was sampled. For this phase a single buffer replay is used and filled before the training with 0.4×10^6 frames using a random policy. The training phase run for for 0.71×10^6 frames and took 5.2 days to complete.

In order to control the exploration-exploitation dilemma it was used an ϵ-greedy policy with ϵ decayed linearly from 1.0 to 0.1 over the first 500k frames and fixed at 0.1 thereafter. The discount factor γ was set to 0.99. As optimizer, the RMSProp algorithm[44] with a batch size of 32 was used. The weights were initialized using the Xavier initialization method.[45]

The DQN algorithm was implemented in Python using the Tensorflow library.[46] Simulations were performed on a workstation with an Intel i7 (8 core) processor, 32 GB of RAM, and the NVIDIA Quadro K2200 as the graphical processing unit.

4.1.2. Results. The results, reported in Figure 7, show that the agent was able to learn an efficient policy for maximizing the reward. The cost decreased stably without any anomaly.

In contrast to the training phase, during which the marker was kept fixed in the same position without any motion, the testing phase was performed positioning the marker on the deck of an unmanned surface vehicle whose motion is affected by different simulated sea conditions. Based on that, four test cases have been defined: (i) static, (ii) rolling, (iii) pitching and (iv) combined. (i) During the static condition, the USV remains still in the same position, without its deck being subject to changes in orientation. (ii) Throughout the rolling condition, the USV oscillated around its x-axis in the range $[-5, 5]$ deg. (iii) In the pitching condition, the USV is subject to perturbations that make it rotating along the y-axis in the range $[-5, 5]$ deg. Finally (iv), for the combined condition the USV's motion was affected on both axis.

The results of the five tests are summarized in Figure 8. The bar chart compares the performance of the trained agent, called DQN-marine, with a random agent, a state-of-the-art AR-Tracker algorithm based on template matching[47] and another network, DQN-multi, previously trained on a different set of textures not including those related to the marine environment.[48]

The average score for the first test (static deck) was 94% for the DQN-marine, 41% for DQN-multi, 100% for the AR-Tracker and 6% for the random agent. The second test (rolling deck) saw a general drop in performance, with the DQN-marine be successful only in 62% of the cases, probably due to the fact the network has been trained on a fixed marker and it never experienced changes of attitude during the training. DQN-multi confirmed a performance of 42%, similar to the previous test. Also the random agent still

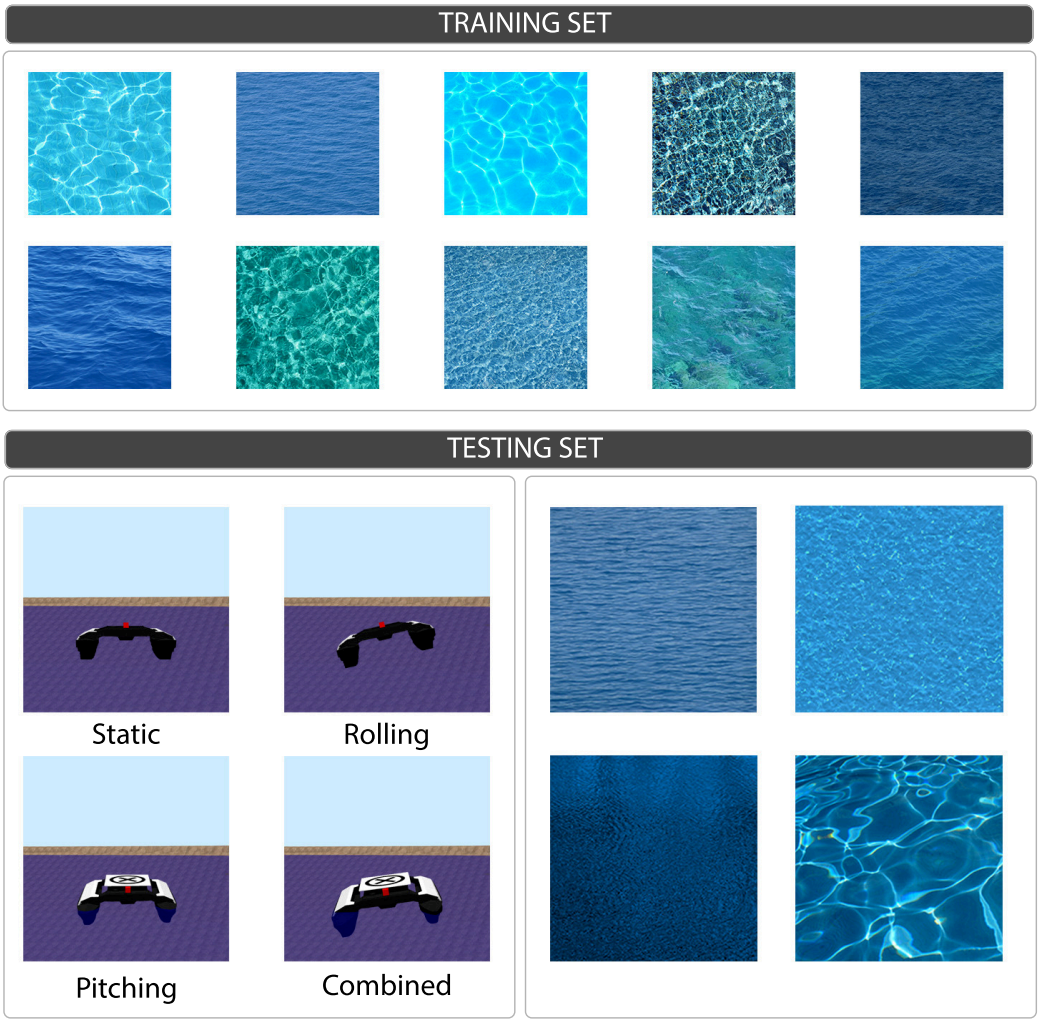

Fig. 5: Textures used during the training (top) and testing (bottom) phases. Four are the conditions in which every agent has been tested: with a static marker, with a rolling deck, a pitching one and when the deck is subject to both rolling and pitching combined.

confirms the same performance (6%). The AR-Tracker proved to still be the agent with the best performance, obtaining a success rate of 96%. In the third test with a pitching deck, the DQN-marine improved its success rate to 98%, as the DQN-multi with 49%. On the other hand, the AR-Tracker saw a drop in performance up to 89%. The random agent has the same success rate of the previous test cases, as expected. The fourth test, combining rolling and pitching, better represents sea conditions. The results show that the DQN-marine is the best agent with performance of 98%, followed by the AR-Tracker with a performance of 94%, the DQN-multi with 43% and the random agent with 6%.

The overall performances are reported in Table I. In general, the AR-Tracker is the best agent in performing the marker detection with a success rate of 95%, followed by the DQN-marine with 88%. This result is drastically influenced by the low performance encountered in the test with a rolling deck. The DQN-multi has the third place with an overall success rate of 44%. This result can be explained saying that the training set used in this work, representing sea surfaces with light reflection and sea foam, presents feature that are different from those in the training set used in Polvara *et al.*[48] As a result, the DQN-multi is not able to generalize well on the new dataset. Lastly, the random agent with 6%. The mean and standard deviation for each group is showed in Table I.

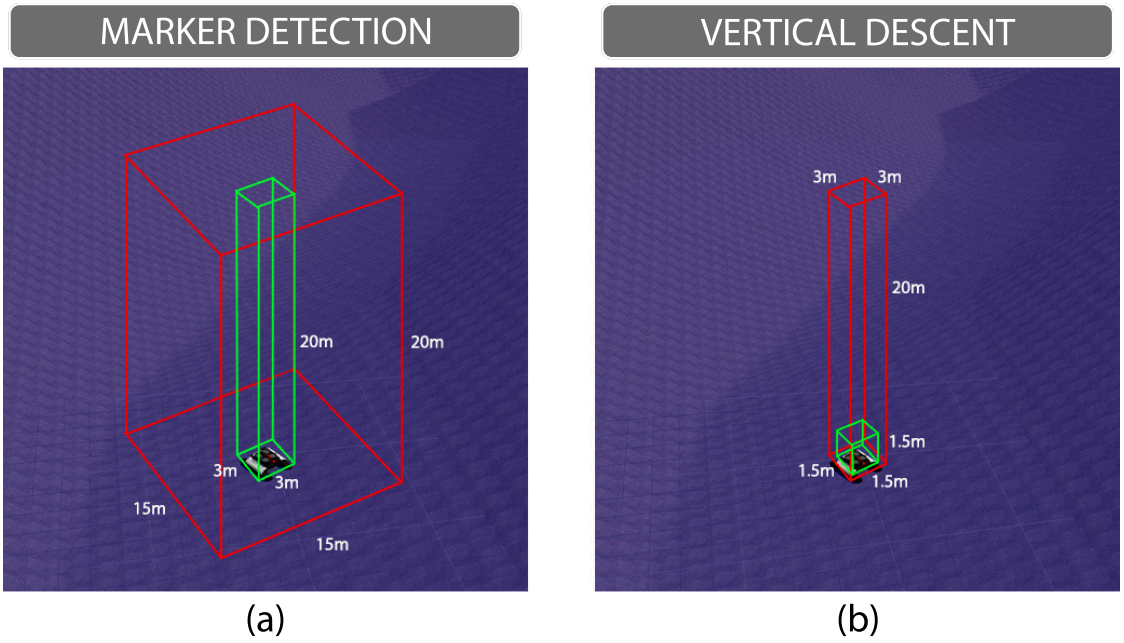

Fig. 6: Bounding boxes for landmark detection (a) and vertical descent (b). The drone is generated in the red box at the beginning of each episode. The green box (target-zone) gives the positive reward.

Table I : Landmark detection success rate (test). Performances obtained by four agents in the marker detection test on four different conditions. The results are in term of mean (standard deviation). The best scores are in bold.

Agent type	**Performance**
DQN-marine	0.88(0.15)
DQN-multi	0.44(0.03)
AR-Tracker	**0.95(0.03)**
Random	0.06(0.01)

It is possible to further analyse the DQN-marine policy observing the action-values distribution in different states (Figure 9). When the UAV is far from the marker, the DQN for landmark detection penalizes the landing action. However, when the drone is over the marker this utility significantly increases triggering the network responsible for the vertical descent.

4.2. Second series of simulation

In the second series of simulations, it was trained and tested the DQN specialized in the vertical descent. To encourage the UAV to descend, during the ϵ-greedy action selection, the action was sampled from a non-uniform distribution where the descend action had a probability of ρ and the other N actions a probability $\frac{1-\rho}{N}$. Moreover, the vehicle was generated at different altitudes ensuring a wider exploration of the state space. Instead of the standard buffer replay, it was used a form of prioritized sampling with three separate buckets for storing neutral, negative and positive experiences, as described in Section 3.3. In this way, a fixed amount of positive and negative experiences was always inserted in the batch at training time.

4.2.1. Methods. During the training phase, the world was represented by a uniform floor of size 100×100 m with the landmark positioned in the centre. The texture was randomly

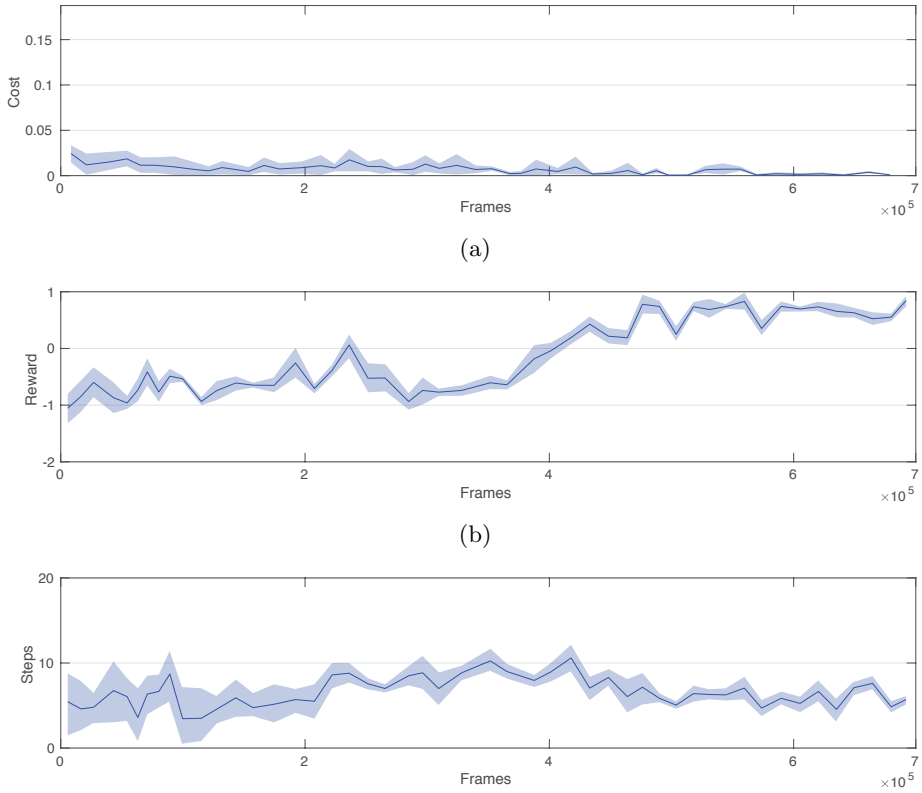

Fig. 7: Results of the first series of simulations. On the x-axis is represented the number of frames, whereas on the y-axis is represented the cost (a), reward (b) and steps per episode (c). The curve shows the average and one standard deviation on five trials.

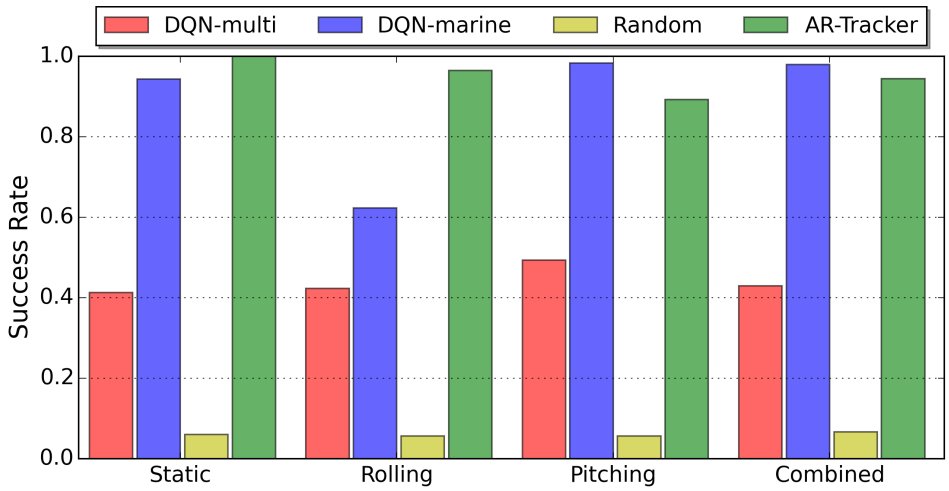

Fig. 8: Comparison of the test phase for marker detection between the DQN trained on marine texture (DQN-marine), the DQN trained on multiple textures (DQN-multi), the AR-Tracker and a random agent.

sampled every 50 episodes among the 10 possible alternatives contained in the training set. The drone could use five actions: forward, backward, left, right, and down. The action was repeated for 2 seconds leading to an approximate shift of 1 m because of the

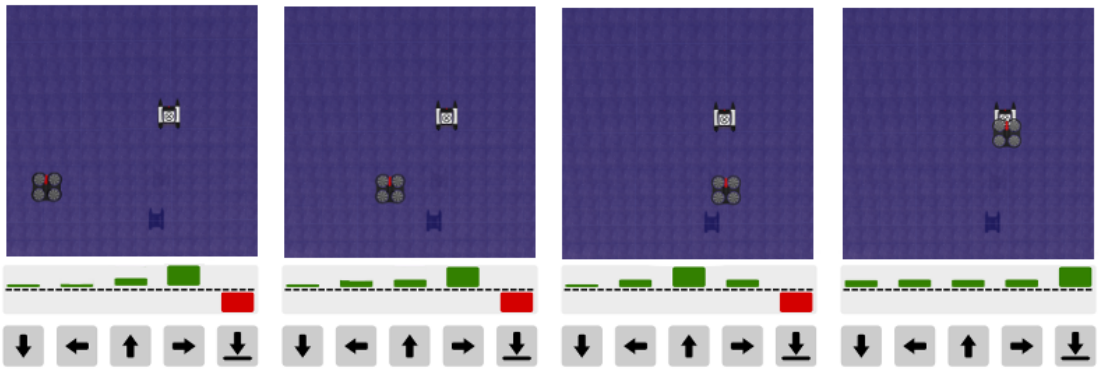

Fig. 9: Snapshots representing the landmark detection. The bottom bar is the utility distribution of the actions. The trigger command has a negative utility (red bar) when the UAV is far from the marker.

constant speed of 0.5 m/s. The descent action was performed at a lower speed of 0.25 m/s to reduce undesired vertical shifts.

The target and policy networks were synchronized every $C = 30000$ frames. For the partitioned buffer replay it was chosen $\rho^+ = 0.25$, $\rho^- = 0.25$, and $\rho^\sim = 0.5$. In this way, a fixed value of 8 positive and negative experiences was always guaranteed in the batch of 32 experiences. A time limit of 80 seconds (40 steps) was used to stop the episode and start a new one. The UAV was generated with a random orientation inside a bounding box of size $3 \times 3 \times 20$ m at the beginning of the episode (Figure 6). A positive reward of 1.0 was given only when the UAV entered in a target-zone of size $1.5 \times 1.5 \times 1.5$ m, centred on the marker. A negative reward of -1.0 was given if the drone descended above 1.5 meter outside the target-zone. A cost of living of -0.01 was applied at each time step.

The same hyper-parameters and hardware configuration described in Section 4.1.1 were used to train the agent. Before the training, the buffer replay was filled using a random policy with 1.2^6 neutral experiences, 2.25×10^5 negative experiences and 2.5×10^5 positive experiences. The number of positive experiences was augmented using horizontal/vertical mirroring and consecutive 90 degrees rotation on all the images stored in the positive partition. This form of data augmentation increased the total number of positive experiences to 5×10^5. The training phase run for for 1.0×10^6 frames and took 8.6 days to complete.

The performance of the agents have been measured as the landing success rate of DQN-marine, DQN-multi, AR-Tracker and random agents in four tests. (i) In the first test the agents landed on the static deck of an USV. (ii) The second test consisted in landing on a rolling deck, while (iii) in the third test the deck was subject to pitching. (iv) In the fourth and last test, landing is accomplished on a deck subject to combined roll and pitch in order to simulate a more realistic condition.

4.2.2. Results. As for the marker detection phase, the results show that the DQN-marine agent was able to learn an efficient policy for maximizing the reward, while the cost decreased stably without any anomaly. In Figure 10 the charts of the cost, reward and steps per episode are illustrated.

The results of the test phase are summarized in Figure 11. The bar chart compares the performances of the DQN-marine, DQN-multi, AR-Tracker and a random agent. The average score for the first test (static deck) is 81% for the DQN-marine, 51% for DQN-multi, 53% for the AR-Tracker and 0% for the random agent. The second test (rolling deck) saw a drop in performance of the DQN-marine to 75%, probably due to the fact the network has been trained on a fixed marker and it never experienced changes of attitude during the training. For the same reason, DQN-multi obtained a lower score

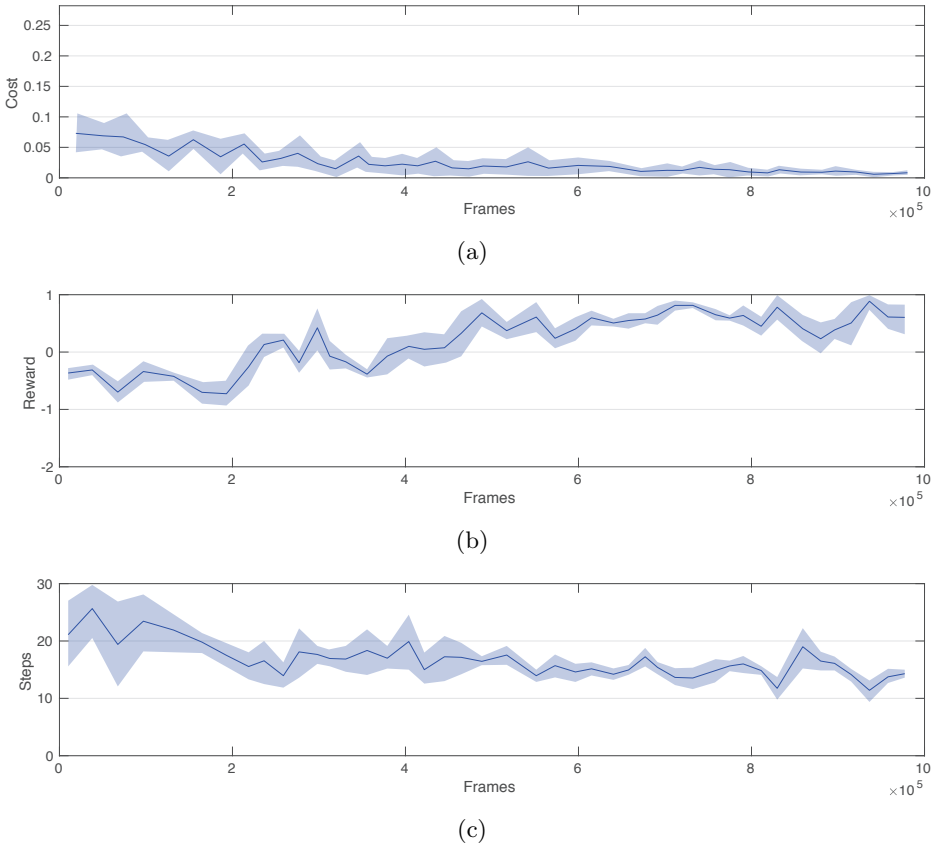

Fig. 10: Results of the second series of simulations. On the x-axis is represented the number of frames, whereas on the y-axis is represented the cost (a), reward (b) and steps per episode (c). The curve shows the average and one standard deviation on five trials.

Table II : Descending manoeuvre success rate (test). Performances obtained by four agents in the descending manoeuvre test on four different conditions. The results are in term of mean (standard deviation). The best scores are in bold.

Agent type	**Performance**
DQN-marine	**0.81(0.06)**
DQN-multi	0.48(0.03)
AR-Tracker	0.64(0.09)
Random	0.02(0.01)

of 45% compared to the previous test. While the random agent still confirms the same null performance, the AR-Tracker proved to be more successful than in the test with a static marker, obtaining now a performance score of 60%. Similar performance have been registered also in the third test with a pitching deck (DQN-marine = 80%, DQN-multi = 47%, AR-Tracker = 71% and random agent = 0%). The fourth test is the one probably most interesting, because better represents situations faced at sea. The results show that the DQN-marine is the best agent with a performance of 79%, followed by the AR-Tracker with 75%, the DQN-multi with 49% and the random agent with null performance. The overall performance grouping all the four test cases are reported in Table II. In general, the DQN-marine is the best agent in the descending manoeuvre with a success rate of 81%, while a state-of-the-art AR-Tracker succeeded only 64% of the times across all the tests performed.

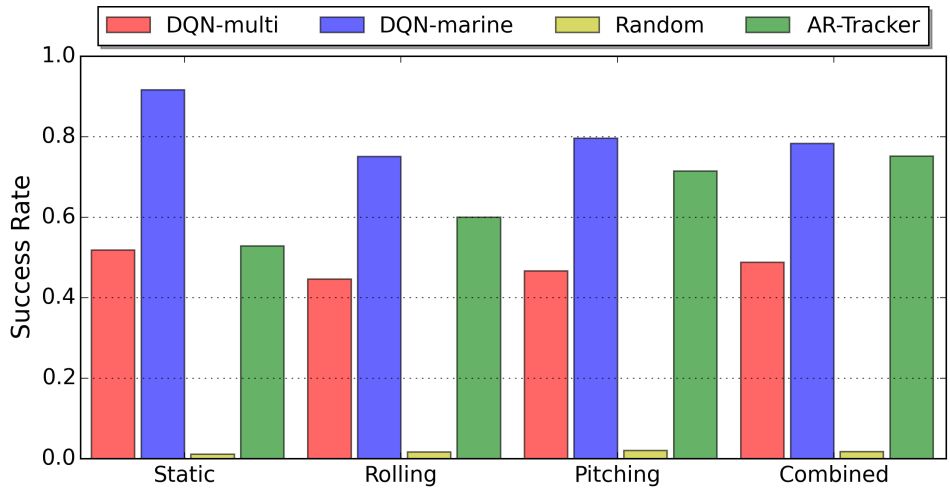

Fig. 11: Comparison of the test phase for the vertical descent between the DQN trained on marine texture (DQN-marine), the DQN trained on multiple textures (DQN-multi), the AR-Tracker and a random agent.

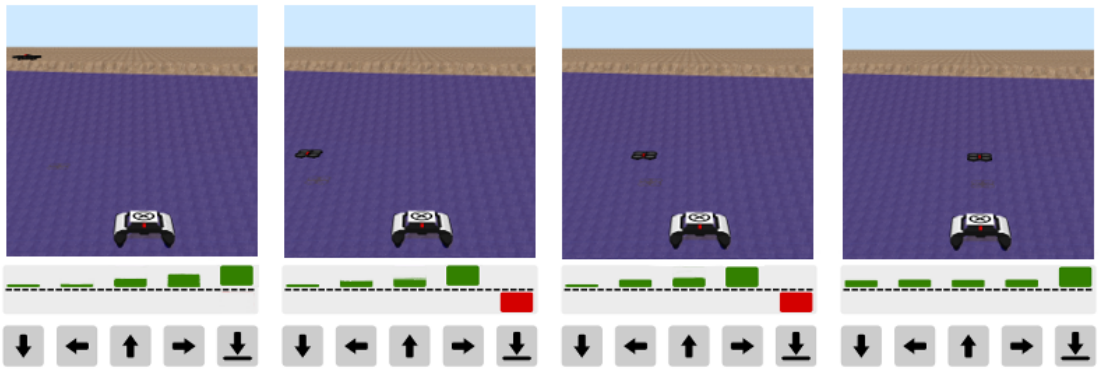

Fig. 12: Snapshots representing the vertical descent. The bottom bar is the utility distribution of the actions. The descent command has a negative utility (red bar) when the drone is close to the ground but far from the marker.

Figure 12 helps to clarify the policy's behaviour. When the UAV is far from the marker, the network encourages the descent. However, when the marker is at the edge of the image acquired by the camera, the utility of lateral movements significantly increases in order to centre the marker again. This procedure continues until the drone reaches the minimum distance required in order to trigger the touchdown phase.

5. Concluding Remarks

In this work, deep reinforcement learning is used to realise a system for the autonomous landing of quad-rotor UAV on the deck of an unmanned surface vehicle that is subject to perturbations induced by sea currents. The main modules of the system are two Deep Q-Networks which can control the UAV in two delicate phases: landmark detection and vertical descent. The two DQNs have been trained in different environments and with different degrees of noise. It has been showed that the system can achieve performances comparable and sometimes even better than traditional state-of-the-art algorithms based on template matching. Moreover, it was showed that it is possible to train robust networks

performing the training on a much simpler condition (like with a static deck), while still obtaining good performance on the real task (rolling and pitching deck).

Future work should focus on two aspects: i) improving performance, and ii) bridging the reality gap. To solve the landing problem it has been decided to use DQN which can be considered a robust approach. However the DRL field is rapidly growing, and new techniques have been recently achieved top performances in complex tasks. A promising technique is Proximal Policy Optimization (PPO)[49] which obtained top results on a large wide variety of continuous motor control problems as well as navigating random 3D mazes, and for this reason it can be particularly suited for the UAV landing problem. Another important point which can contribute to obtain better performances is an efficient exploration of the state space. Exploration has ever been a major challenge in reinforcement learning. Shallow methods such as ϵ-greedy could not be the best choice for temporally extended exploration. For instance, some modern versions of Thompson sampling such as posterior sampling[50] or bootstrapped DQN[51] could reduce the convergence time required to obtain stable policies.

The system has been tested only in a simulated environment and moving to real world application could be challenging because of the reality gap. The reality gap is the obstacle that makes difficult to implement many robotic solutions in real world. This is especially true for DRL where a large number of episodes is necessary in order to obtain stable policies. Thanks to a technique known as domain randomization,[52] a method for training models on simulated images that transfer to real images by randomizing rendering in the simulator, we have already been able to accomplish autonomous landing on a fixed marker in the real world. But the marine environment poses many more challenges than an indoor one, therefore further studies must be completed before proceeding with a real implementation. For example, these studies should take care of modelling adverse wind and light conditions (such as reflection in the water) at training time. Also sea currents modelling can be further improved to be more realistic than the one used in this actual work. As result, if on one hand the training time could increase due to a more complex environment (leading to multiple different features that need to be extracted and learnt), on the other one the gap existing between the simulator and the real world decreases making the adoption of this end-to-end method in the real world even more feasible.

In conclusion, the results obtained are promising and show that it is possible to apply DQN to complex problems such as landing on the perturbed deck of a USV. However, further research is necessary in order to reach stable policies which can effectively work in a wide range of conditions (e.g., mutable lights, sea and wind currents etc).

References

1. Y. Altshuler, V. Yanovsky, I. A. Wagner, and A. M. Bruckstein, "Efficient cooperative search of smart targets using uav swarms," *Robotica*, vol. 26, no. 4, p. 551557, 2008.
2. R. Polvara, S. Sharma, R. Sutton, J. Wan, and A. Manning, "Toward a multi-agent system for marine observation," in *Advances in Cooperative Robotics*. World Scientific, 2017, pp. 225–232.
3. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
4. S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.
5. W. Kong, D. Zhou, D. Zhang, and J. Zhang, "Vision-based autonomous landing system for unmanned aerial vehicle: A survey," in *Multisensor Fusion and Information Integration for Intelligent Systems (MFI), 2014 International Conference on*. IEEE, 2014, pp. 1–8.
6. A. Gautam, P. Sujit, and S. Saripalli, "A survey of autonomous landing techniques for uavs," in *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*. IEEE, 2014, pp. 1210–1218.
7. J. E. Gomez-Balderas, S. Salazar, J. A. Guerrero, and R. Lozano, "Vision-based autonomous hovering for a miniature quad-rotor," *Robotica*, vol. 32, no. 1, p. 4361, 2014.
8. S. Saripalli and G. Sukhatme, "Landing on a moving target using an autonomous helicopter," in *Field and service robotics*. Springer, 2006, pp. 277–286.
9. B. Herissé, T. Hamel, R. Mahony, and F.-X. Russotto, "Landing a vtol unmanned aerial vehicle on a moving platform using optical flow," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 77–89, 2012.

10. A. Cesetti, E. Frontoni, A. Mancini, P. Zingaretti, and S. Longhi, "A vision-based guidance system for uav navigation and safe landing using natural landmarks," in *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009.* Springer, 2009, pp. 233–257.
11. F. Ruffier and N. Franceschini, "Optic flow regulation in unsteady environments: a tethered mav achieves terrain following and targeted landing over a moving platform," *Journal of Intelligent & Robotic Systems*, vol. 79, no. 2, pp. 275–293, 2015.
12. J. Kim, Y. Jung, D. Lee, and D. H. Shim, "Outdoor autonomous landing on a moving platform for quadrotors using an omnidirectional camera," in *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on.* IEEE, 2014, pp. 1243–1252.
13. O. A. Yakimenko, I. I. Kaminer, W. J. Lentz, and P. Ghyzel, "Unmanned aircraft navigation for shipboard landing using infrared vision," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, no. 4, pp. 1181–1200, 2002.
14. K. E. Wenzel, P. Rosset, and A. Zell, "Low-cost visual tracking of a landing place and hovering flight control with a microcontroller," *Journal of Intelligent & Robotic Systems*, vol. 57, no. 1, pp. 297–311, 2010.
15. C. Theodore, D. Rowley, A. Ansar, L. Matthies, S. Goldberg, D. Hubbard, and M. Whalley, "Flight trials of a rotorcraft unmanned aerial vehicle landing autonomously at unprepared sites," in *Annual Forum Proceedings-American Helicopter Society*, vol. 62, no. 2. AMERICAN HELICOPTER SOCIETY, INC, 2006, p. 1250.
16. S. Scherer, L. Chamberlain, and S. Singh, "Autonomous landing at unprepared sites by a full-scale helicopter," *Robotics and Autonomous Systems*, vol. 60, no. 12, pp. 1545–1562, 2012.
17. M. Sereewattana, M. Ruchanurucks, and S. Siddhichai, "Depth estimation of markers for uav automatic landing control using stereo vision with a single camera," in *Int. Conf. Information and Communication Technology for Embedded System.*, 2014.
18. H. Shi and H. Wang, "A vision system for landing an unmanned helicopter in a complex environment," in *Sixth International Symposium on Multispectral Image Processing and Pattern Recognition.* International Society for Optics and Photonics, 2009, pp. 74 962G–74 962G.
19. O. Shakernia, Y. Ma, T. J. Koo, and S. Sastry, "Landing an unmanned air vehicle: Vision based motion estimation and nonlinear control," *Asian journal of control*, vol. 1, no. 3, pp. 128–145, 1999.
20. J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.
21. Y. Cui, T. Matsubara, and K. Sugimoto, "Kernel dynamic policy programming: Applicable reinforcement learning to robot systems with high dimensional states," *Neural Networks*, 2017.
22. J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.
23. J. A. Bagnell and J. G. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 2. IEEE, 2001, pp. 1615–1620.
24. A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," *Experimental Robotics IX*, pp. 363–372, 2006.
25. T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on.* IEEE, 2016, pp. 528–535.
26. H. M. Fayek, M. Lech, and L. Cavedon, "Evaluating deep learning architectures for speech emotion recognition," *Neural Networks*, 2017.
27. M. Patacchiola and A. Cangelosi, "Head pose estimation in the wild using convolutional neural networks and adaptive gradient methods," *Pattern Recognition*, 2017.
28. M. S. Pavel, H. Schulz, and S. Behnke, "Object class segmentation of rgb-d video using recurrent convolutional neural networks," *Neural Networks*, vol. 88, pp. 105–113, 2017.
29. J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.
30. S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, p. 0278364917710318, 2016.
31. V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.
32. X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks." in *Aistats*, vol. 15, no. 106, 2011, p. 275.
33. P. WawrzyńSki and A. K. Tanwani, "Autonomous reinforcement learning with experience replay," *Neural Networks*, vol. 41, pp. 156–167, 2013.
34. H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in *AAAI*, 2016, pp. 2094–2100.
35. S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

36. T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.
37. K. Narasimhan, T. Kulkarni, and R. Barzilay, "Language understanding for text-based games using deep reinforcement learning," *arXiv preprint arXiv:1506.08941*, 2015.
38. A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.
39. M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
40. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.
41. N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2149–2154.
42. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.
43. J. Dias, K. Althoefer, and P. U. Lima, "Robot competitions: What did we learn?[competitions]," *IEEE Robotics & Automation Magazine*, vol. 23, no. 1, pp. 16–18, 2016.
44. T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, 2012.
45. X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks." in *Aistats*, vol. 9, 2010, pp. 249–256.
46. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.
47. R. Polvara, S. Sharma, J. Wan, A. Manning, and R. Sutton, "Vision-based autonomous landing of a quadrotor on the perturbed deck of an unmanned surface vehicle," *Drones*, vol. 2, no. 2, p. 15, 2018.
48. R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi, "Autonomous quadrotor landing using deep reinforcement learning," *arXiv preprint arXiv:1709.03339*, 2017.
49. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
50. I. Osband, D. Russo, and B. Van Roy, "(more) efficient reinforcement learning via posterior sampling," in *Advances in Neural Information Processing Systems*, 2013, pp. 3003–3011.
51. I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," in *Advances in Neural Information Processing Systems*, 2016, pp. 4026–4034.
52. J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *arXiv preprint arXiv:1703.06907*, 2017.